Automated deployment of enterprise systems in large-scale environments

Takoua Abdellatif[1], Didier Hoareau[2], and Yves Mahéo[2]

[1] BULL SA, LSR-IMAG / INRIA,
Takoua.Abdellatif@inrialpes.fr
[2] Valoria Lab – University of South Brittany
{Didier.Hoareau|Yves.Maheo}@univ-ubs.fr

Abstract. The deployment of multi-tiered applications in large-scale environments remains a difficult task: the architecture of these applications is complex and the target environment is heterogeneous, open and dynamic. In this paper, we show how the component-based approach simplifies the design, the deployment and the reconfiguration of a J2EE system. We propose an architecture description language that allows specifying constraints on the resources needed by the components and on their location, and a deployment solution that handles failures.

Introduction J2EE application servers are complex service-oriented architectures. They are generally deployed on clusters to improve their quality of service. A J2EE cluster is composed of replicated Web and EJB tiers for load balancing and fault tolerance. A front-end load balancer dispatches the HTTP requests to the containers. In large-scale environments, machines are highly distributed and heterogeneous in terms of software and hardware configurations. Furthermore, these resources can be dynamic. Therefore the resource allocation should be automated and the deployment process should automatically take into account the dynamicity of the environment. We make the assumption that the large-scale environment is structured in zones and that for each zone are defined some known machines called *zone managers* whose role is to maintain a list of the machines in the zone and to orchestrate the deployment process.

Deployment system We adopt an architecture-based approach to manage the J2EE system. We wrap system parts into explicitly bound components. The obtained system, called *JonasALaCarte*, is based on the Fractal component model.

In large-scale environments, we cannot know in advance the target machine for each component of the system. So, in order to specify the deployment of a J2EE system, we have added to the Fractal architecture descriptor (that defines the architecture of the system in terms of component definitions and component bindings) a *deployment descriptor*, that contains, for each component, the description of the resources that the target platform must satisfy, and references to component instances. The deployment descriptor lists all the constraints that a hosting machine has to verify. *Resource constraints* allow hardware and software needs to be represented, and *location constraints* make it possible to control the placement of a component when more than one host apply for its hosting. These constraints are solved thanks to our deployment process that allows, additionally, the recovery from failures.

Deployment process For each zone in the environment, a zone manager maintains the list of the machines in the zone that may host the J2EE system components. This zone manager is given the architecture and deployment descriptors by an administrator. It then multicasts them to the zone nodes. Each node checks the compatibility of its local resources with the resources required for each component. If it satisfies all the resource constraints associated with a component, it sends to the manager its candidature for the instantiation of this component. The manager receives several candidatures and tries to compute a placement solution in function of the location constraints and the candidatures. The manager updates the deployment descriptor with the new placement information and broadcasts it to all the zone nodes. Each node that receives the new deployment descriptor updates its own one and is thus informed of which components it is authorized to instantiate and of the new location of the other components. The final step consists in downloading necessary packages from well defined package repositories whose location is defined in the deployment descriptor.

The steps described above define a *propagative deployment*, that is, necessary components for running J2EE applications can be instantiated and started without waiting for the deployment of all the components. As soon as a resource become available or a machine offering new resources enters the network, candidatures for the installation of the "not yet installed" components will make the deployment progress.

Some preliminary experiments we have conducted on a prototype implementation show that the performance of the resource observation and the constraint solving remain acceptable even for a large number of non trivial resource constraints.

Automatic recovery from failures In the environment we target, resources can also become unavailable, some parts of the J2EE system can be faulty and some machine may fail. In this work, we consider silent failures. When a component does not respond to a method call or a request within a timeout, the node detecting the failure sends to the zone manager a message holding the identity of the component to redeploy. Then, the zone manager updates the deployment descriptor by removing the location of the component and broadcasts the new descriptor to all the machines connected in the zone. This automates the redeployment of the faulty component since all the machines find themselves back in the propagative deployment described above.

Specific actions are carried out in the case of the failure of replicated components (eg EJB container and Web container services) or zone managers, exploiting group communication and temporary storage of incoming requests.

Conclusion The work described in this paper proposes a solution for the deployment of enterprise systems in large-scale environments. Our main contribution consists in the following points. First, the deployment system is resource-aware and the constraint resolution is performed in a reasonable time. Second, the deployment task is simplified since the administrator role is reduced to writing a deployment descriptor. All the deployment process and the recovery from failures are automated. Finally, we maintain the performability of the system since we maintain the structure described in the architecture descriptor by replacing each time a faulty component by another. This allows assuring the continuity of Internet services and maintaining their quality of service.